"Express Mail" mailing label number:

EV324252832US

INSTRUCTION SAMPLING IN A MULTI-THREADED PROCESSOR

Adam R. Talcott

Mario I. Wolczko

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to processors, and more particularly to sampling mechanisms of processors.

Description of the Related Art

One method of understanding the behavior of a program executing on a processor is for a processor to randomly sample instructions as the instructions flow through the instruction pipeline. For each sample, the processor gathers information about the execution history and provides this information to a software performance monitoring tool. Unlike tools which aggregate information over many instructions (e.g., performance counters), such an instruction sampling mechanism allows the performance analyst to map processor behaviors back to a specific instruction.

In a multi-threaded processor, the processor's architectural state is replicated for each thread. For example, each thread has its own copy of the general purpose registers. This leads to much replication of state, at commensurate implementation cost.

SUMMARY OF THE INVENTION

In accordance with the invention, particular properties of instruction sampling are used to reduce the implementation cost of sampling in a multi-threaded processor by avoiding unnecessary replication. More specifically, the present invention allows software using a sampling mechanism to specify which samples (such as which

threads) are of interest and to allow the sampling mechanism to discard or ignore uninteresting samples. Such a method accomplishes sampling in a multithreaded processor without replicating sampling hardware for each thread of the processor.

One aspect of a sampling mechanism in accordance with the present invention is that the data provided by the sampling mechanism represents a statistically valid set of samples, but are not, by definition, a complete representation of all instructions executed. The sampling mechanism described in this invention exploits this feature to avoid the cost and complexity of replicating the sampling mechanism for each thread (namely, the tagging, recording and reporting facilities); instead the sampling mechanism provides a single mechanism which is shared by all threads, and multiplexed between them. The sampling mechanism samples instructions without initial regard to the thread. When an instruction is tagged, the thread of that instruction is noted. When the instruction sample is complete, the sample is matched against per-thread filtering criteria; the matching is the only part of the mechanism which is replicated for each thread.

This approach for sampling within a multi threaded processor provides a cheaper and simpler implementation of sampling. An additional advantage is that the distribution of samples matches the distribution of fetch slots used by each thread. A sampling mechanism which is replicated for each thread would not have this distribution.

In another embodiment, multithreaded sampling may be implemented with even less hardware replication by providing a single set of hardware filtering criteria, and having the operating system schedule the sampling mechanism among threads. This implementation saves hardware cost, but precludes fairness among threads. This implementation might be more desirable for a processor which includes a relatively larger number of threads (e.g., a processor which supports eight threads per processor core).

The sampling mechanism can be enabled independently for each thread. Each thread has its own candidate counter and sample selection criteria registers. However, there is a single set of sample instruction history registers shared across threads. The

sampling logic samples a single instruction at a time, alternating between threads. When a sampled instruction becomes a reporting candidate, the per-thread candidate counter is decremented; if the candidate counter becomes zero, a sample trap pending signal is asserted and the sample is reported via a disrupting trap to the thread from which it was sampled. Sampling for both threads is disabled until software resets the sample trap pending signal.

Sampling may be restarted only after the trap handler has copied the contents of the history registers. Hence, it is desirable for the trap handler to accomplish the copying with minimal delay; until the copying is completed, neither thread can sample instructions (because the history registers could be overwritten prematurely).

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention may be better understood, and its numerous objects, features and advantages made apparent to those skilled in the art by referencing the accompanying drawings. The use of the same reference number throughout the several figures designates a like or similar element.

Figure 1 shows a block diagram of a processor having a sampling mechanism in accordance with the present invention.

Figures 2A and 2B show a flow chart of the operation of the sampling mechanism in accordance with the present invention.

DETAILED DESCRIPTION

Referring to Figure 1, processor 100 includes sampling mechanism 102. This sampling mechanism 102 is provided to collect detailed information about individual instruction executions. The sampling mechanism 102 is coupled to the instruction fetch unit 110 of the processor 100. The fetch unit 110 is also coupled to the remainder of the processor pipeline 112. Processor 100 includes additional processor elements as is well known in the art.

The sampling mechanism 102 includes sampling logic 120, instruction history registers 122, sampling registers 124, sample filtering and counting logic 126 and notification logic 128. The sampling logic 120 is coupled to the instruction fetch unit 110, the sampling registers 124 and the sample filtering and counting logic 126. The instruction history registers 122 receive inputs from the instruction fetch unit 110 as well as the remainder of the processor pipeline 112; the instruction history registers 122 are coupled to the sampling registers 124 and the sample filtering and counting logic 126. The sampling registers 124 are also coupled to the sample filtering and counting logic 126. The sample filtering and counting logic 126 are coupled to the notification logic 128.

The sampling mechanism 102 collects detailed information about individual instruction executions. If a sampled instruction meets certain criteria, the instruction becomes a reporting candidate. When the sampling mode is enabled, an instruction is selected randomly by the processor 100 (via, e.g., a linear feedback shift register) as the instructions are fetched. An instruction history is created for the selected instruction. The instruction history is a collection of information including such things as events induced by the sample instruction and various associated latencies. When all events for the sample instruction have been generated (e.g., after the instruction retires or aborts), the vector of events gathered by the instruction history is compared with a user supplied vector, which indicates the events of interest.

In one embodiment, software indicates the properties or events of interest via a bit vector contained in a register. Each bit in this vector corresponds to a property or event which the hardware can gather for an instruction sample. The register can be used as a filter which hardware can apply to an instruction sample to determine if that sample is a candidate for reporting to software. Once these properties or events have been specified, the hardware mechanism that gathers samples compares each sample against the vector of desired properties or events. Hardware can determine a match by combining the software-specified filter with an instruction sample. This combination could be a simple mask operation, or a more expressive operation. Based upon the comparison, the hardware may reject the sample without incurring any software overhead. If the sample matches one or more software-specified properties or events, the sample can be reported to software.

The instruction history including program counter (PC), and privilege status are examples of criteria used in selecting whether an instruction may become a candidate for sampling. If an instruction meets the eligibility test and becomes a candidate, a candidate counter is decremented. If the counter becomes zero, the instruction sample is reported via the notification logic 128. Software copies the instruction's history from the instruction history registers 122 and resets the candidate counter.

Figures 2A and 2B show a flowchart of the operation of sampling mechanism 102. More specifically, at step 210, the software sets filtering criteria and loads a candidate counter register, located within the sample filtering and counting logic 126, with a non-zero value, thus enabling the sampling logic 120. Once the counter register is loaded, the sample filtering and counting logic 126 delays sampling by a random number of cycles at step 222. Next the fetch unit 110 selects a random instruction from a current fetch bundle at step 224. The instruction is analyzed to determine whether a valid instruction has been selected at step 226. If not, then the sampling mechanism 102 returns to step 222.

If the fetched instruction is a valid instruction, then instruction information is captured at step 230. The instruction information includes, for example, the program counter (PC) of the instruction as well as privileged information and context information of the instruction. Next, the sampling logic 120 clears the instruction history registers 122 at step 232. Next, during execution of the instruction by the processor 100, the sampling logic 120 gathers events, latencies, etc. for the sampled instruction at step 234. The sampling logic 120 then reviews the processor state to determine whether all possible events for the selected instruction have occurred at step 236. If not, then the sampling logic 120 continues to gather events etc. at step 234.

If all possible events for the selected instruction have occurred, then the instruction is examined at step 240 to determine whether the selected instruction matches the filtering criteria (i.e., is the selected instruction of interest to the software?). One of the filtering criteria against which the selected instruction is compared is the thread to which the selected instruction is bound. If not, then control

returns to step 222 where the counting logic 126 delays the sampling by a random number of cycles to select another instruction for sampling.

If yes, then the counting logic 126 decrements a candidate counter at step 244. Next the candidate counter is analyzed to determine whether the candidate counter is zero at step 246. If the candidate counter is not zero, then control returns to step 222 where the counting logic 126 delays the sampling by a random number of cycles prior to selecting another instruction. If the candidate counter equals zero, then the notification logic 128 reports the sampled instruction at step 248. The candidate counter register value is used to count candidate samples which match the selection criteria. On the transition from 1 to 0 (when made by hardware following a sample) a notification is provided and the instruction history is made available via the instruction history registers. The counter then stays at zero until changed by software. The power-on value of the candidate counter register value is 0. The candidate counter allows software to control how often samples are reported, and thus limits the reporting overhead for instructions which are both interesting and frequent. The software then processes the sampled instruction history at step 250 and the processing of the sampling mechanism 102 finishes.

The sampling mechanism 102 functions in a multithreaded processor. The sampling mechanism 102 is enabled independently for each thread of the processor 100. Each thread of the processor has its own candidate counter and selection criteria registers. However, there is a single set of history registers shared across threads. The sampling logic 120 samples a single instruction at a time, alternating between threads. When a sampled instruction becomes a reporting candidate, the per-thread candidate counter is decremented; if the per-thread candidate counter becomes zero, the sample is reported to the thread from which it was sampled. Sampling is restarted only after the contents of the history registers has been copied and sampling is enabled. Accordingly, it is desirable to accomplish this task with minimal delay; until the copying has been completed, neither thread can have instructions sampled (because the history registers could be overwritten prematurely).

The present invention is well adapted to attain the advantages mentioned as well as others inherent therein. While the present invention has been depicted,

described, and is defined by reference to particular embodiments of the invention, such references do not imply a limitation on the invention, and no such limitation is to be inferred. The invention is capable of considerable modification, alteration, and equivalents in form and function, as will occur to those ordinarily skilled in the pertinent arts. The depicted and described embodiments are examples only, and are not exhaustive of the scope of the invention.

For example, while the sampling mechanism 102 is shown coupled to the instruction fetch unit 110, it will be appreciated that the sampling mechanism 102 may be coupled to any location in the processor in which instruction information could be sampled.

Also for example, while certain sample selection criteria values and sample instruction history values have been set forth, it will be appreciated that any combination of these values as well as other values are within the scope of the invention.

Also for example, the above-discussed embodiments include modules that perform certain tasks. The modules discussed herein may include hardware modules or software modules. The hardware modules may be implemented within application specific circuitry or via some form of programmable logic device. The software modules may include script, batch, or other executable files. The modules may be stored on a machine-readable or computer-readable storage medium such as a disk drive. Storage devices used for storing software modules in accordance with an embodiment of the invention may be magnetic floppy disks, hard disks, or optical discs such as CD-ROMs or CD-Rs, for example. A storage device used for storing firmware or hardware modules in accordance with an embodiment of the invention may also include a semiconductor-based memory, which may be permanently, removably or remotely coupled to a microprocessor/memory system. Thus, the modules may be stored within a computer system memory to configure the computer system to perform the functions of the module. Other new and various types of computer-readable storage media may be used to store the modules discussed herein. Additionally, those skilled in the art will recognize that the separation of functionality into modules is for illustrative purposes. Alternative embodiments may merge the

functionality of multiple modules into a single module or may impose an alternate decomposition of functionality of modules. For example, a software module for calling sub-modules may be decomposed so that each sub-module performs its function and passes control directly to another sub-module.

Consequently, the invention is intended to be limited only by the spirit and scope of the appended claims, giving full cognizance to equivalents in all respects.